STRUTS2 LOCALIZATION, INTERNATIONALIZATION (I18N)

Internationalization (i18n) is the process of planning and implementing products and services so that they can easily be adapted to specific local languages and cultures, a process called localization. The internationalization process is sometimes called translation or localization enablement. Internationalization is abbreviated **i18n** because the word starts with an i and ends with an n, and there are 18 characters between the first i and the last n.

Struts2 provides localization ie. internationalization (i18n) support through resource bundles, interceptors and tag libraries in the following places:

- The UI Tags
- Messages and Errors.
- Within action classes.

Resource Bundles:

Struts2 uses resource bundles to provide multiple language and locale options to the users of the web application. You don't need to worry about writing pages in different languages. All you have to do is to create a resource bundle for each language that you want. The resource bundles will contain titles, messages, and other text in the language of your user. Resource bundles are the file that contains the key/value pairs for the default language of your application.

The simplest naming format for a resource file is:

```
bundlename_language_country.properties
```

Here **bundlename** could be ActionClass, Interface, SuperClass, Model, Package, Global resource properties. Next part **language_country** represents the country locale for example Spanish (Spain) locale is represented by es_ES and English (United States) locale is represented by en_US etc. Here you can skip country part which is optional.

When you reference a message element by its key, Struts framework searches for a corresponding message bundle in the following order:

- ActionClass.properties
- Interface.properties
- SuperClass.properties
- model.properties
- package.properties
- struts.properties
- global.properties

To develop your application in multiple languages, you would have to maintain multiple property files corresponding to those languages/locale and define all the content in terms of key/value pairs. For example if you are going to develop your application for US English (Default), Spanish, and Franch the you would have to create three properties files. Here I will use **global.properties** file only, you can make use of different property files to segregate different type of messages.

- **global.properties:** By default English (United States) will be applied
- **global_fr.properties:** This will be used for Franch locale.
- **global_es.properties:** This will be used for Spanish locale.

Access the messages:

There are several ways to access the message resources, including getText, the text tag, key attribute of UI tags, and the i18n tag. Let us see them in brief:

To display **i18n** text, use a call to **getText** in the property tag, or any other tag, such as the UI tags as follows:

```
<s:property value="getText('some.key')" />
```

The **text tag** retrieves a message from the default resource bundle ie. struts.properties

```
<s:text name="some.key" />
```

The **i18n tag** pushes an arbitrary resource bundle on to the value stack. Other tags within the scope of the i18n tag can display messages from that resource bundle:

```
<s:i18n name="some.package.bundle">
     <s:text name="some.key" />
</s:i18n>
```

The **key** attribute of most UI tags can be used to retrieve a message from a resource bundle:

```
<s:textfield key="some.key" name="textfieldName"/>
```

Localization Example:

Let us target to create **index.jsp** from the previous chapter in multiple languages. Same file would be written as follows:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Form with Multilingual Support</title>
</head>

<body>
   <h1><s:text name="global.heading"/></h1>

   <s:url  >
      <s:param name="request_locale" >en</s:param>
   </s:url>
   <s:url  >
      <s:param name="request_locale" >es</s:param>
   </s:url>
   <s:url  >
      <s:param name="request_locale" >fr</s:param>
   </s:url>

   <s:a href="%{indexEN}" >English</s:a>
   <s:a href="%{indexES}" >Spanish</s:a>
   <s:a href="%{indexFR}" >France</s:a>

   <s:form action="empinfo" method="post" namespace="/">
```

```
      <s:textfield name="name" key="global.name" size="20" />
      <s:textfield name="age" key="global.age" size="20" />
      <s:submit name="submit" key="global.submit" />
   </s:form>

</body>
</html>
```

We will create **success.jsp** file which will be invoked in case defined action returns SUCCESS.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Success</title>
</head>
<body>
   <s:property value="getText('global.success')" />
</body>
</html>
```

Here we would need to create following two actions. (a) First action a to take care of Locale and display same index.jsp file with different language (b) Another action is to take care of submitting form itself. Both the actions will return SUCCESS, but we will take different actions based on return values because our purpose is different for both the actions:

Action to take care of Locale:

```
package com.tutorialspoint.struts2;

import com.opensymphony.xwork2.ActionSupport;

public class Locale extends ActionSupport{
   public String execute()
   {
       return SUCCESS;
   }
}
```

Action to submit the form:

```
package com.tutorialspoint.struts2;

import com.opensymphony.xwork2.ActionSupport;

public class Employee extends ActionSupport{
   private String name;
   private int age;

   public String execute()
   {
       return SUCCESS;
   }

   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }
   public int getAge() {
       return age;
   }
```

```
   public void setAge(int age) {
       this.age = age;
   }
}
```

Now. let us create following three **global.properties** files and put the in CLASSPATH:

global.properties:

```
global.name = Name
global.age = Age
global.submit = Submit
global.heading = Select Locale
global.success = Successfully authenticated
```

global_fr.properties:

```
global.name = Nom d'utilisateur
global.age = l'âge
global.submit = Soumettre des
global.heading = Sé lectionnez Local
global.success = Authentifi é  avec succès
```

global_es.properties:

```
global.name = Nombre de usuario
global.age = Edad
global.submit = Presentar
global.heading = seleccionar la configuracion regional
global.success = Autenticado correctamente
```

We will create our **struts.xml** with two actions as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
   <constant name="struts.devMode" value="true" />
   <constant name="struts.custom.i18n.resources" value="global" />

   <package name="helloworld" extends="struts-default" namespace="/">
      <action name="empinfo"

         method="execute">
         <result name="input">/index.jsp</result>
         <result name="success">/success.jsp</result>
      </action>

      <action name="locale"

         method="execute">
         <result name="success">/index.jsp</result>
      </action>
   </package>

</struts>
```

Following is the content of **web.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee"
```

```
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
   >

   <display-name>Struts 2</display-name>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>

   <filter>
      <filter-name>struts2</filter-name>
      <filter-class>
         org.apache.struts2.dispatcher.FilterDispatcher
      </filter-class>
   </filter>

   <filter-mapping>
      <filter-name>struts2</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
</web-app>
```

Now, right click on the project name and click **Export > WAR File** to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start Tomcat server and try to access URL http://localhost:8080/HelloWorldStruts2/index.jsp. This will give you following screen:

Now select any of the languages, let us say we select **Spanish**, it would display the following result:

You can try with Franch language as well. Finally, let us try to click **Submit** button when we are in Spanish Locale, it would display the following screen:

Congratulations, now you have a multi-lingual webpage, you can launch your website globally.